Data Structure and Algorithms Technical Report
Name : Melissa Sim Rui Qi
Student ID : 22515845
Language used for the assessment : Java

1. Overview

The objective of this project is to develop a modular delivery optimisation system that optimises its delivery network in a metropolitan area. The system is designed to handle route planning, customer lookup, scheduling parcel and sorting of delivery records using data structure and algorithms.

1.1 Summary of my solution

This project is divided into four modules, Module 1 focusing on Gtaph-Based Route Planning, I used adjacency list to represent the delivery route, Breath-First Search (BFS) is used to find the reachable areas while Depth-First Search (DFS) is used for detecting cycles in the delivery network and a function that calculates the shortest path to find the best delivery route.

Module 2 focuses on Hash-Based Customer Lookup, I implemented a hash table using linear probing and double hashing to store and retrieve customer's information using customer IDs and to prevent collision.

Module 3 focuses on Heap-Based Parcel Scheduling using max heap structure to prioritize the deliveries based on the calculated priority time. The function inserts the delivery status with "In Transit" and "Delayed" into the heap using the insert function and the extract_priority function retrieves the top value in the heap structure and removes it.

Module 4 focuses on Sorting Delivery Records using a custom sorting algorithm (Quick Sort Median of Three and Merge Sort Recurse).

Each of this module was designed to meet the project requirements.

1.2 Modular Design

This program meets the modular design principles by having high cohesion where each of the module focuses on one specific task. Although low coupling is not achieved throughout the entire project but the principle is still met in some of the modules. Information hiding is also met by hiding its internal details and only exposing only what others need and reuse of codes is also met where shared functionality is in one place reused throughout the project wherever it is needed.

2. Data Structure Used

The data structure used in this project are graph, hash table, heap and sorting algorithm. The purpose for using graph for route planning is to represent the delivery network of the path between the starting point and destination. This is because it supports fast insert of edges and vertices with its weight (delivery time) and it is also an ideal choice for graph traversing such as Breath-First Search (BFS) for reachability and Depth-First Search for detecting cycles. A custom shortest path calculation is also implemented to find the best delivery route.

The purpose of using hash table for customer lookup is to store and retrieve customer's information using customer IDs. This is useful for fast customer lookup during delivery or scheduling deliveries. Linear probing and double hashing is used for this project, o ensure there is an efficient use of space and to handle collisions.

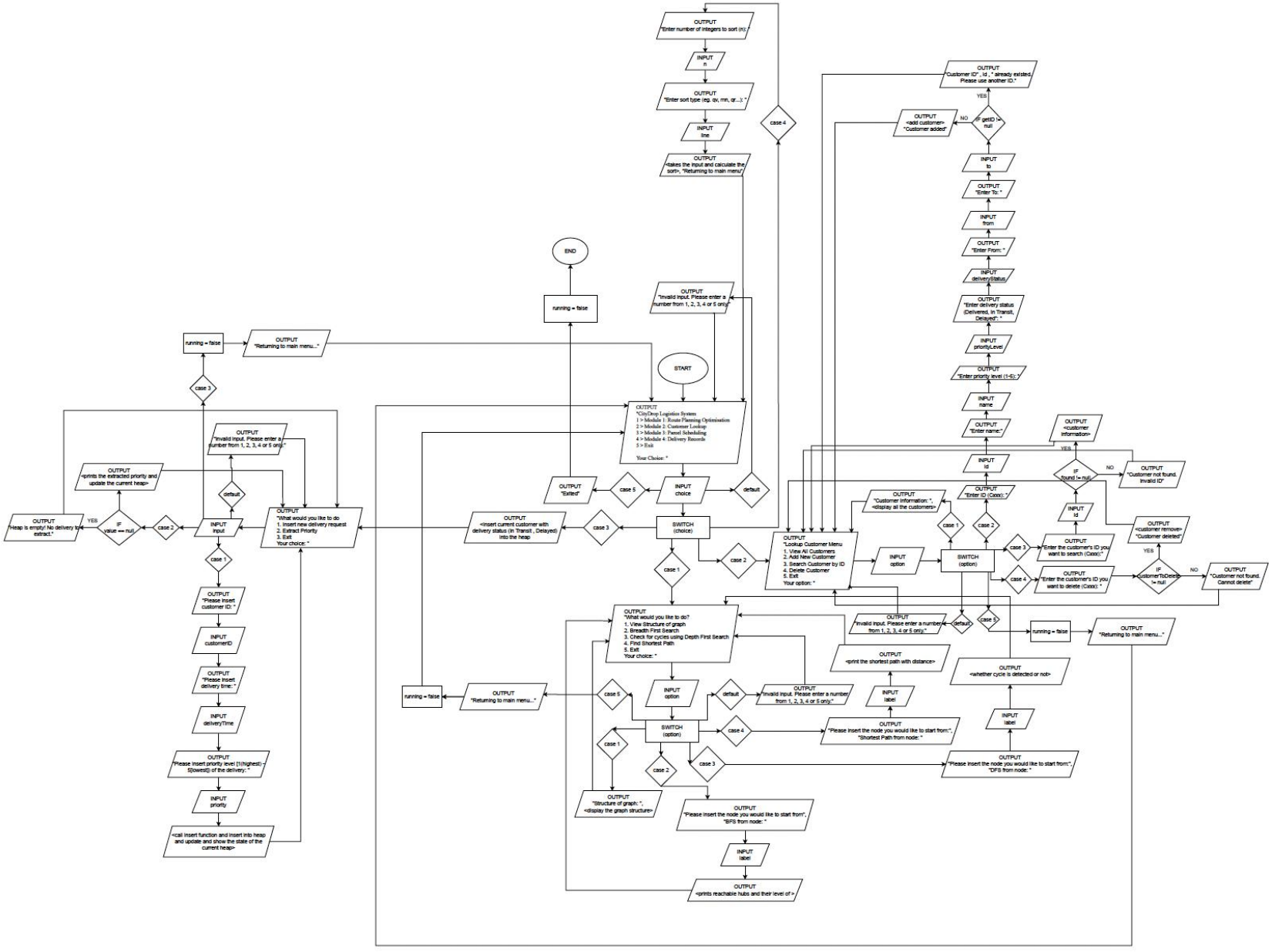
The purpose of using heap for parcel scheduling is to prioritize deliveries based on the customer's priority and delivery time. A max heap is used for this to allow efficient access to the highest priority parcel in the heap. This heap structure also supports insertion and removal operations.

Lastly, the purpose of using sorting algorithm is to sort records for the deliveries based on the estimated delivery time. A custom sort is implemented such as merge sort and quick sort, which gives off a easier readability and analysis of the delivery data.

3. Module Integration

The flowchart below shows how the program runs, we will first read all the necessary input files, then performs each module based on the user's selection. Module 1 allows the user to have a look at the delivery routes and the shortest path to get from a specific starting point to their destination. Module 2 lets the user to view all the customers, add and delete customer. Then, module 3 will let the user to view the delivery priorities and insert customer if needed. Lastly, module 4 is to sort the delivery records based on the estimated delivery time.

<The original file for flowchart is included in the .zip file just in case the one below is blurry>


START
END
OUTPUT
"CityDrop Logistics System
1 > Module 1: Route Planning Optimisation
2 > Module 2: Customer Lookup
3 > Module 3: Parcel Scheduling
4 > Module 4: Delivery Records
5 > Exit
Your Choice: "
INPUT
choice
SWITCH
(choice)
case 1
case 2
case 3
case 4
case 5
default
OUTPUT
"Invalid input. Please enter a number from 1, 2, 3, 4 or 5 only."
OUTPUT
"Exited"
running = false
OUTPUT
"Enter number of integers to sort (n):"
INPUT
n
OUTPUT
"Enter sort type (eg. qv, mn, qr...): "
INPUT
line
OUTPUT
<takes the input and calculate the sort>, "Returning to main menu"
OUTPUT
"Lookup Customer Menu
1. View All Customers
2. Add New Customer
3. Search Customer by ID
4. Delete Customer
5. Exit
Your option: "
INPUT
option
SWITCH
(option)
OUTPUT
"Customer Information: ",
<display all the customers>
OUTPUT
"Enter ID (Cxxx): "
INPUT
id
IF
found != null
NO
OUTPUT
"Customer not found.
Invalid ID"
YES
OUTPUT
<customer
information>
OUTPUT
"Enter the customer's ID you
want to search (Cxxx):"
OUTPUT
"Enter the customer's ID you
want to delete (Cxxx): "
IF
customerToDelete
!= null
OUTPUT
<customer remover>
"Customer deleted"
OUTPUT
"Customer not found.
Cannot delete"
OUTPUT
"Enter name:"
INPUT
name
OUTPUT
"Enter priority level (1-5): "
INPUT
priorityLevel
OUTPUT
"Enter delivery status
(Delivered, In Transit,
Delayed): "
INPUT
deliveryStatus
OUTPUT
"Enter From: "
INPUT
from
OUTPUT
"Enter To: "
INPUT
to
OUTPUT
<add customer>
"Customer added"
IF getID !=
null
OUTPUT
"Customer ID" , id , " already existed.
Please use another ID."
OUTPUT
"Returning to main menu..."
OUTPUT
"What would you like to do
1. Insert new delivery request
2. Extract Priority
3. Exit
Your choice: "
OUTPUT
<insert current customer with
delivery status (In Transit , Delayed)
into the heap
OUTPUT
"Please insert
customer ID: "
INPUT
customerID
OUTPUT
"Please insert
delivery time: "
INPUT
deliveryTime
OUTPUT
"Please insert priority level (1(highest)-
5(lowest)) of the delivery: "
INPUT
priority
<call insert function and insert into heap
and update and show the state of the
current heap>
INPUT
input
IF
value == null
YES
OUTPUT
"Heap is empty! No delivery to
extract."
OUTPUT
<prints the extracted priority and
update the current heap>
OUTPUT
"What would you like to do?
1. View Structure of graph
2. Breadth First Search
3. Check for cycles using Depth First Search
4. Find Shortest Path
5. Exit
Your choice: "
OUTPUT
"Structure of graph: ",
<display the graph structure>
OUTPUT
"Please insert the node you would like to start from",
"BFS from node: "
OUTPUT
"Please insert the node you would like to start from",
"DFS from node: "
OUTPUT
"Please insert the node you would like to start from:",
"Shortest Path from node: "
OUTPUT
<print the shortest path with distance>
OUTPUT
<whether cycle is detected or not>
INPUT
label
OUTPUT
<prints reachable hubs and their level of >
OUTPUT
"Returning to main menu..."

4. Algorithm Complexity

Modules	Time	Space
1 > Breath-First Search	O(n)	O(n)
1 > Depth-First Search	O(n)	O(n)
1 > Shortest Path	O(n^2)	O(n)
2 > Hash Table (insert/search)	O(1)	O(n)
3 > Heap (insert/extract)	O(log n)	O(n)
4 > Merge Sort (Recurse)	O(n log n)	O(n)
4 > Quick Sort (Median of three)	Average: O(n log n) Worst: O(n^2)	O(log n)

4.1 Performance for Module 4

4.1.1 Reversed

Datasets	Merge Sort Recurse	Quick Sort Median of Three
100	471.0	106.0
500	50.0	579.667
1000	82.667	1129.667

4.1.2 Random Order

Datasets	Merge Sort Recurse	Quick Sort Median of Three
100	23.0	65.667
500	674.0	632.333
1000	553.0	66.0

4.1.3 Nearly Sorted

Datasets	Merge Sort Recurse	Quick Sort Median of Three
100	15.0	5.0
500	1198.667	21.0
1000	68.667	30.667

4.2 Discussion of Performance

Based on the performance, Quick Sort Median of Three is a better choice, with a guaranteed time complexity of O(n log n). However, merge sort recurse performs better on revered and random data. Both have an average complexity of O(n log n), however Quick Sort is preferred for average case speed in this project.

5. Sample Output

5.1 Module 1

5.1.1 Structure of Graph

This shows the adjacency list of each nodes together with its weight (estimated travel time).

```
Structure of graph:
A -> B(4 minutes) C(2 minutes) D(7 minutes) G(9 minutes)
B -> A(4 minutes) C(3 minutes) D(3 minutes) E(6 minutes) F(7 minutes)
C -> A(2 minutes) B(3 minutes) F(5 minutes)
D -> A(7 minutes) B(3 minutes) E(4 minutes) F(10 minutes)
E -> B(6 minutes) D(4 minutes) F(2 minutes)
F -> B(7 minutes) C(5 minutes) E(2 minutes) G(1 minutes) D(10 minutes)
G -> F(1 minutes) A(9 minutes)
H ->
```

5.1.2 Breath First Search

Using the node given, this will show the list of reachable nodes with levels, node H is not in the list because it is a disconnected node.

```
Please insert the node you would like to start from:
BFS from node: A
A - Level 0
B - Level 1
C - Level 1
D - Level 1
G - Level 1
E - Level 2
F - Level 2
```

5.1.3 Check for Cycles

Using the node given, this will show whether there is any cycle detected from that node. Since node H is a disconnected node, it does not have any cycles.

```
Please insert the node you would like to start from:
DFS from node: A
Cycle detected!
```

```
Please insert the node you would like to start from:
DFS from node: H
No cycle detected.
```

5.1.4 Shortest Path

This will show the calculated shortest path from the node given to every other node.

```
Please insert the node you would like to start from:
Shortest Path from node: A
Shortest distance from A:
A = 0
A -> B = 4
A -> C = 2
A -> D = 7
A -> C -> F -> E = 9
A -> C -> F = 7
A -> C -> F -> G = 8
A -> H = No path
```

5.2 Module 2

Collision Handling :

```
Collision detected for key C025 at index 27 resolved to index 30
Collision detected for key C026 at index 28 resolved to index 31
Collision detected for key C027 at index 29 resolved to index 32
Collision detected for key C028 at index 30 resolved to index 33
Collision detected for key C029 at index 31 resolved to index 34
Collision detected for key C054 at index 25 resolved to index 35
Collision detected for key C055 at index 26 resolved to index 36
Collision detected for key C054 at index 30 resolved to index 40
Collision detected for key C055 at index 31 resolved to index 41
Collision detected for key C014 at index 3 resolved to index 9
Collision detected for key C015 at index 4 resolved to index 10
Collision detected for key C045 at index 0 resolved to index 11
Collision detected for key C046 at index 1 resolved to index 12
Collision detected for key C047 at index 2 resolved to index 13
Collision detected for key C035 at index 66 resolved to index 75
Collision detected for key C036 at index 67 resolved to index 76
Collision detected for key C037 at index 68 resolved to index 77
Collision detected for key C038 at index 69 resolved to index 78
Collision detected for key C039 at index 70 resolved to index 79
Collision detected for key C044 at index 96 resolved to index 14
```

View All Customer (show some only for the report):

```
------------------------------
Customer ID: C043
Name: Luke
Address: 1034 Oak Street
Priority Level: 2
Delivery Status: In Transit
From: D
To: G
------------------------------
------------------------------
Customer ID: C010
Name: Jack
Address: 112 Willow Street
Priority Level: 5
Delivery Status: Delayed
From: D
To: F
------------------------------
```

Adding new customer with collision handling

```
Enter ID (Cxxx): C999
Enter name: Melissa
Enter address: Pickles Land
Enter priority level (1-5): 1
Enter delivery status (Delivered, In Transit, Delayed): Delivered
Enter From: D
Enter To: G

Collision detected for key C999 at index 78 resolved to index 80
Customer added.
```

Finding customer that exist and not
[Exist]

```
Enter the customer's ID you want to search (Cxxx): C045
------------------------------
Customer ID: C045
Name: Nate
Address: 1056 Chestnut Boulevard
Priority Level: 4
Delivering From: C
Delivering To: D
Delivery Status: Delivered
------------------------------
```

[Does Not Exist]

```
Enter the customer's ID you want to search (Cxxx): C49
Customer not found. Invalid ID.
```

Deleting customer that exist and not
[Exist]

```
Enter the customer's ID you want to delete (Cxxx): C999
Customer deleted
```

[Does Not Exist]

```
Enter the customer's ID you want to delete (Cxxx): C111
Customer not found. Cannot delete.
```

5.3 Module 3

Inserting from sample input file, Customer whose parcel is already delivered will be skipped, not inserting into the heap.

```
Delivery Status: Delivered
Skip -- Customer ID: C045

Delivery Status: In Transit
Trying to insert to heap: Customer ID: C046 | C >> E
Delivery time: 7

Priority level for C046: 143

State of current heap:
Heap priorities: [143]
```

Inserting new delivery request
[Before]

```
State of current heap:
Heap priorities: [1004 , 505 , 504 , 337 , 504 , 202 , 336 , 169 , 255 , 145 , 503 , 145 , 171 , 167 , 167 , 113 , 1
44 , 146 , 146 , 112 , 127 , 128 , 167 , 112 , 143 , 143 , 168 , 146 , 147]
```

[After - current heap is updated with the highest priority on the top]

```
Please insert customer ID: C999
Please insert delivery time: 1
Please insert priority level [1(highest) -5(lowest)] of the delivery: 1

Priority level for C999: 1005

State of current heap:
Heap priorities: [1005 , 505 , 1004 , 337 , 504 , 202 , 504 , 169 , 255 , 145 , 503 , 145 , 171 , 167 , 336 , 113 ,
144 , 146 , 146 , 112 , 127 , 128 , 167 , 112 , 143 , 143 , 168 , 146 , 147 , 167]
```

Extracting Priority

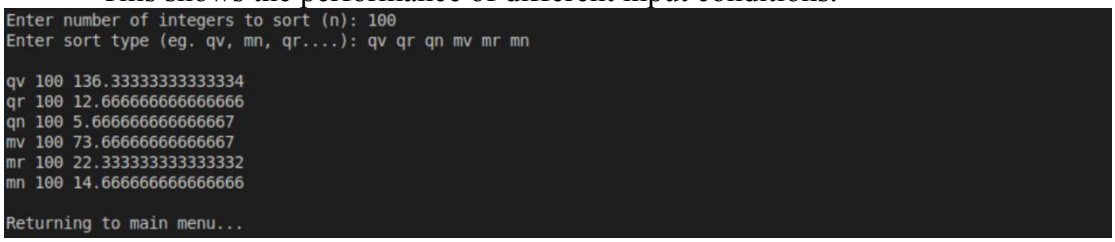
```
Prioriy extracted:
Customer ID : C999
Delivery Time: 1
Priority Level: 1

State of current heap:
Heap priorities: [1004 , 505 , 504 , 337 , 504 , 202 , 336 , 169 , 255 , 145 , 503 , 145 , 171 , 167 , 167 , 113 , 1
44 , 146 , 146 , 112 , 127 , 128 , 167 , 112 , 143 , 143 , 168 , 146 , 147]
```

The highest priority is extracted and the next highest will be moved towards the top of the heap.

5.4 Module 4

This shows the performance of different input conditions.



6. Reflection

Throughout this assignment, I learned how different data structures such as graphs. heap, hash tables and sorting algorithm are applied in the real world systems. This project also helped deepened my understanding on this unit's concept. Although I faced some challenges along the way, but it helps me think critically and search fro solutions, which increased my knowledge on this unit. Overall, this assignment particularly increased my understanding of hash tables and heaps.

7. Limitations and Assumptions

For the customer ID, I assumed that each customer can only make one delivery, meaning the customer cannot have the same ID with different priorities and delivery schedules.

For module 1 and module 2, the sample input file's name are all hard-coded, however if I have enough time, I would like it to be passing an argument into it rather than hard-coding the file's name, this way the program can support sample input files with different names.